COBB

INSIDE VISUAL dBASE™

Tips & techniques for Visual dBASE and dBASE for Windows

June 1996 • Vol. 3 No. 6
US $8.50

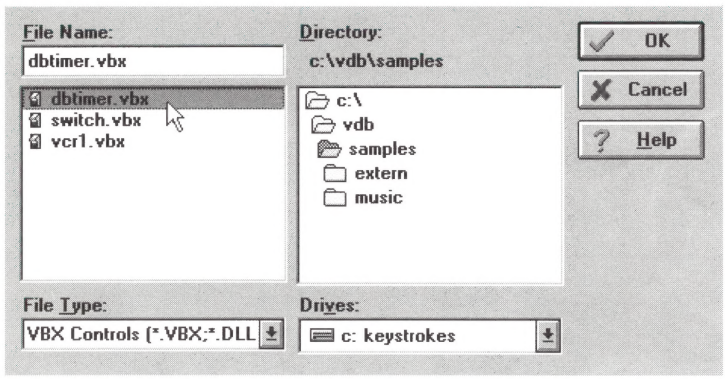
Visual dBASE Specific

A tale of two tickers

by Keith Chuvala

Sometimes, it makes good sense to control the length of time a user can keep a form open onscreen or how long a status message remains visible before it changes to a warning message. Visual dBASE provides two mechanisms for triggering actions at specified time intervals: the DBTimer Visual BASIC (VBX) control and the timer class. In this article, we'll demonstrate both mechanisms and discuss situations in which you might want to use each one.

Figure A



You select the DBTIMER.VBX control to add it to the Form Designer's Control Palette.

IN THIS ISSUE

DBTimer: A form-based timer

The DBTimer VBX is part of the standard Visual dBASE installation and should be visible in the Form Designer's Control Palette on the VBX page. If you don't have a DBTimer on the Control Palette, you'll need to add it manually.

While in the Form Designer, select Setup Custom Controls from the File menu. Click the VBX Controls tab to display currently installed VBX controls. If you don't see *DBTimer* in the list, click the Add... button, then select the DBTIMER.VBX file, as illustrated in **Figure A**. When you click the OK button, the DBTimer control icon will appear in the Form Designer's Control Palette, as shown in **Figure B** on the next page.

VBX controls exhibit a number of interesting—some might say *odd*—behaviors and characteristics. One of the oddities is that Visual dBASE's context menus, which you normally access by right-clicking on a given control, don't work when you position the mouse over a VBX control. If the Inspector window isn't currently displayed, you must click on some other control or on the form itself to select the Inspector, in order to bring the control into view. Then, you must click on the VBX control to display its properties in the Inspector window.

The DBTimer control responds to only three events: OnDesignOpen, OnOpen, and OnTimer. OnTimer dictates what will happen when the specified time elapses. This event can take the form of a dBASE procedure or function name, or it can be a code block.

The DBTimer control also has more than a dozen properties, but only two are unique and important to its function—Enabled and Seconds. The *Enabled* property is a logical value that dictates whether the timer control

is active. The Seconds property contains the interval of time that will pass before the code specified in the OnTimer event handler executes.

Test-driving DBTimer

As a first experiment with the DBTimer control, we'll make a form that beeps every two seconds. Begin by creating a form and dragging a DBTimer control to it. It doesn't matter where you position the control on the form, since it doesn't appear when the form runs. The control icon simply gives you something to click on while you're working in Design mode. (That's why we can safely ignore most of the control's properties—they deal with position, dimensions, and such.)

Specify the following values for the new DBTimer control in the Inspector:

```
OnTimer = { ; ?? chr(7) }
Seconds = 2
Enabled = .T.
```

Save the form, then look at the code produced by the Form Designer, as shown in Listing A. You'll note a couple of lines necessary for VBX support that you don't normally see. First, the command

```
load dll
```

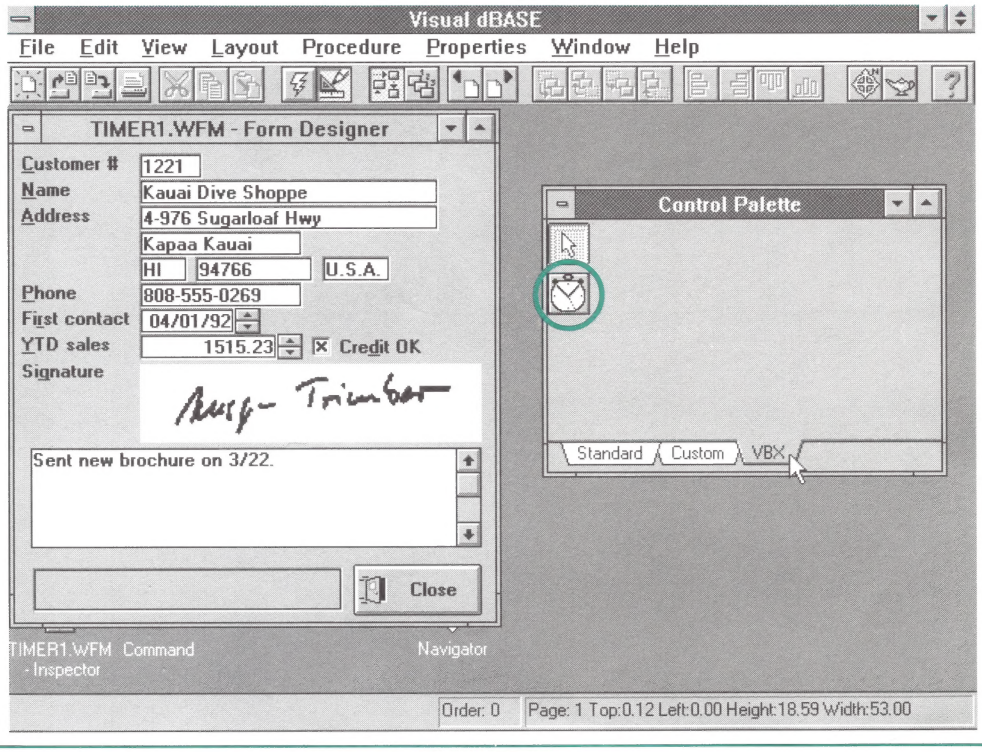
loads the VBX control into memory. The VBStream property of the control itself references a binary file that dBASE needs at runtime in order to work with the VBX. If you create executable programs for distribution with Visual dBASE, you'll need to pay attention to these two items, since you must include the VBX file itself and its BFM file with the runtime support files for your application.

Run the form, and the speaker will beep every two seconds until you either close or release the form—or until the noise drives you batty and you smash the computer. You can manage the DBTimer control programmatically as the form is running by changing the values of one of these properties: Seconds, Enabled, or OnTimer.

The Timer class

Visual dBASE also sports a Timer class that doesn't require the external files you need

Figure B



The DBTimer control icon now appears in the Control Palette.

for a VBX. Timer isn't a documented class, but it's been around since dBASE for Windows 5.0. Because it's undocumented, there's no guarantee that it will exist in its current form in future upgrades of Visual dBASE. A built-in class, Timer also has no visual element, so it doesn't appear in the Control Palette of the Form Designer. Therefore, you must program it in a form's OnOpen event handler or some other appropriate spot. Fortunately, using the Timer class is just about as easy as using the DBTimer control, and you don't have to put Timer in a form to work with it.

From the Command window, the following commands will produce the same annoying beeping we just created via the VBX control:

```
tBeep = new Timer()
tBeep.Interval = 2
tBeep.Enabled = .t.
tBeep.OnTimer = {; ?? chr(7) }
```

The only significant syntactical difference between the Timer class and a DBTimer control is that the Timer class uses *Interval* instead of *Seconds* to specify the amount of time to be counted between OnTimer events.

The operational differences between the two are more significant. While you create and release a DBTimer along with the form you place it on, an instance of the Timer class exists from the time you create it to the time you release it—events that might or might not be associated with form behaviors.

The best of both tools?

An ideal timer control would contain the best attributes of both the Timer class and a DBTimer control. Such a timer control would be a pseudo-visual control, like DBTimer, that you could drop on a form. The ideal timer wouldn't require the external files necessary for the VBX control. And, finally, you could change the ideal timer's settings either initially in the Form Designer or programmatically at runtime.

The good news is that you can, in fact, write a control that accomplishes all those goals; well, almost all of them. The custom code defined by the code in **Listing B** is a good start.

Let's take a look at this approach. By using inheritance to obtain all the features of the timer class, this simple custom control does everything a Timer object does. It's simple, straightforward, and easy to understand. As good as it sounds, this approach is faulty. You can't drop it on a form, even though it will show up on the Control Palette's Custom tab. The keyword Custom identifies the code as a custom control, but a custom control must be based on a visual component in order for you to be able to use it in the Form Designer.

Listing A: *Form code with a DBTimer on board*

```
** END HEADER - do not remove this line*
* Generated on 03/20/96
*
parameter bModal
local f
f = new DBTDEMOFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS DBTDEMOFORM OF FORM
   this.Top     = 5
   this.Text    = "DBTimer Demo"
   this.Height = 10
   this.Width  = 40
   this.Left    = 10

   load dll C:\VDB\SAMPLES\DBTIMER.VBX
   DEFINE DBTIMER DBTIMER1 OF THIS;
       PROPERTY;
         Top 3.6465,;
         Height 1.6475,;
         Width 4.667,;
         VBStream "C:\VDB\SAMPLES\DBTDEMO.BFM 56",;
         Left 16.666,;
         OnTimer {; ?? chr(7)}

ENDCLASS
```

Listing B: *An almost custom control*

```
* Timer1.CC
*
class Timer1 of Timer custom
   this.Interval = 10 && default interval
   this.Enabled  = .f. && default state
   this.OnTimer  = {; ?? chr(7)}  && default action
endclass
```

The code in Listing C hooks a Timer object to a PaintBox control. We could choose any of the built-in controls as a starting point, but the PaintBox is designed for this kind of control, and it provides a bare-bones control framework that does nothing on its own to disturb our environment.

The code from this control is more complex than our first attempt. Since the control is no longer based on the Timer class, we must create the Timer itself at runtime. We've added procedures for setting up, enabling, and disabling the timer programmatically.

Whither OnRightDblClick?

The oddest thing you'll find in the code for this control is that the OnTimer event handler is attached to the OnRightDblClick event, which occurs when the user double-clicks the right mouse button. (Visual dBASE doesn't let us create our own events—they're inherited from the base class, which, in this case, is PaintBox. Therefore, we chose an event that we wouldn't normally use in an application.) You rarely need to write a program that uses the OnRightDblClick event. However, if for some reason you need to use that event in your application, you might pick another obscure event for your custom timer control.

When you drop a copy of this new timer on a form, its display is pretty dull—a PaintBox has no particular display attributes, and it looks like a rectangle with no border. Figure C shows a form in Design mode with the Timer2 control in place. Remember, although this control isn't particularly attractive in Design mode, it must have some visual component to facilitate selecting it while in Design mode. Besides, the control will be invisible at runtime, so users will never know just how ugly it is.

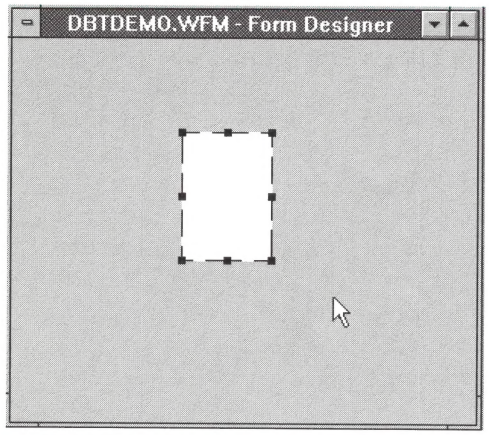
Even with all that new work, this control fails to live up to our ideal timer's specifications: There's a problem with the Interval property. You can set that property programmatically, but you can't set it in the Form Designer. However, this limitation isn't our fault!

Listing C: *A Timer based on PaintBox*

```
* Timer2.CC
*
class Timer2(f) of PaintBox(f) custom
   this.interval = 10
   this.visible  = .f.
   this.name     = "Timer2"
   this.OnOpen   = Class::Setup

   Procedure Setup
      parameters interval,ontimer
      if type("this.timer") <> "O"
         this.timer = new timer()
         this.timer.enabled = this.enabled
      endif
      if type("interval") == "N"
         this.SetInterval(interval)
      endif
      if type("this.OnRightDblClick") $ "FP CB"
         this.SetTimer(this.OnRightDblClick)
      endif
      if this.enabled
         this.enable()
      endif

   Procedure SetInterval
      parameters nInterval
      if type("nInterval") == "N"
         this.interval = nInterval
      endif

   Procedure SetTimer
      parameters fpTimer
      if type("fpTimer") $ "FP CB"
         this.ontimer = fpTimer
      endif

   Procedure Enable
      if type("this.timer") <> "O"
         this.timer = new timer()
      endif
      this.timer.ontimer  = this.OnRightDblClick
      this.timer.enabled = .t.

   Procedure Disable
      if type("this.timer") <> "O"
         this.timer = new timer()
      endif
      this.enabled = .f.
      this.timer.enabled = .f.

endclass
```

In its current incarnation, Visual dBASE doesn't properly record changes made to custom properties. (Borland refers to this as "streaming out custom properties.") You can change the values of custom properties using the Inspector window in the Form Designer, but when the form's code is written to disk, the original value of each custom property is recorded instead of any changes you might have made. So, our Interval property is useless in the Form Designer.

Figure C



The Timer2 control, based on a PaintBox, appears in Design mode as a plain rectangle on the form.

The workaround

Borland has acknowledged the problem with changes to custom properties and will, we hope, fix this anomaly in a future upgrade. In the meantime, we must work around this problem. Remember, the Form Designer *does* record inherited properties and any changes made to them in the Inspector Window.

With this in mind, **Listing D** makes use of the control's Width property to hold the interval value. Is this an ugly workaround? You bet. This solution requires documentation so you won't forget that width is being used for a time value. But this solution works fine, and if using the Width property doesn't suit your style or standards, you can pick any other numeric control inherited from PaintBox. ❖

Listing D: *A fully functional Timer control*

```
* MyTimer.CC
*
class MyTimer(f) of PaintBox(f) custom
   this.interval = 10
   this.width    = 10   && Holds the REAL Interval
   this.visible  = .f.
   this.name     = "MyTimer"
   this.OnOpen   = Class::Setup

   Procedure Setup
      parameters interval,ontimer
      if type("this.timer") <> "O"
         this.timer = new timer()
         this.timer.enabled = this.enabled
      endif
      if type("interval") == "N"
         this.SetInterval(interval)
      endif
      if type("this.OnRightDblClick") $ "FP CB"
         this.SetTimer(this.OnRightDblClick)
      endif
      if this.enabled
         this.enable()
      endif

   Procedure SetInterval
      parameters nInterval
      if type("nInterval") == "N"
         this.width = nInterval
         this.interval = nInterval
      endif

   Procedure SetTimer
      parameters fpTimer
      if type("fpTimer") $ "FP CB"
         this.ontimer = fpTimer
      endif

   Procedure Enable
      if type("this.timer") <> "O"
         this.timer = new timer()
      endif
      this.timer.interval = this.width
      this.timer.ontimer  = this.OnRightDblClick
      this.timer.enabled = .t.

   Procedure Disable
      if type("this.timer") <> "O"
         this.timer = new timer()
      endif
      this.enabled = .f.
      this.timer.enabled = .f.

endclass
```

Report Tip

Formatting report page numbers to make them easier to find

When you use Crystal Reports for dBASE to create a report, both the Expert and the Designer give you the option of numbering the pages. Automatic pagination saves time when you create a standard report. However, by default the program numbers the report pages in the bottom-left corner of the page, as shown in **Figure A**.

Unfortunately, this page-numbering style may prevent the reader from seeing the page number at first glance. In this article, we'll show you how to make your reports easier to read by adding special formatting to the page numbers and by labeling the page numbers with the word *Page*, as shown in **Figure B**.

Formatting the page number

Crystal Reports for dBASE uses PageNumber—a predetermined field—to number report pages automatically. If you don't use one of the experts to create the report for you, you can manually place the PageNumber field almost anywhere in your Page header or Page footer band. To do so, just open the Insert menu, choose the Special Field option, and then select Page Number.

Next, drag the field icon to the spot where you want the page number to appear and click once. To apply formatting to the page number using the Format menu, simply select the PageNumber field and open the Format menu or right-click

Figure A

SALES
4/26/96

LNAME	FNAME	MKTCODE
Jones	Peter	85
Barnes	Peter	85
Noble	Peter	85
Greer	David	85
Guy	Mark	85
Tonnes	Jesus	85
Johnson	Emmanel	85
Rolfson	Samuel	85
Washington	Joe	85
Gorman	Billy	85
Richardson	Bobby	85
Sawyer	Bob	85
Jackson	Rhett	85
Ramirez	Robin	85
Gonzalez	Pat	8203
Knoll	Terry	96
Davis	Sam	96
Schultz	Sandy	96
Chandler	Rick	96
Donaldson	Peter	96
Williamson	Tony	96
Frederickson	Peter	96
Johnson	Larry	96
Flintstone	Fred	96
Rubble	Peter	96
Hackett	Sonya	96
Stein	Gerry	96
Cline	Geoffrey	96
Klein	Linda	96
Boswell	Janie	96
Lowell	Janine	96
Adams	Lois	96
Boston	Betty	96
Carr	Sandra	96
Kerr	Stanley	96
Dautel	Jennifer	96
Taft	Stan	96
Kennedy	Jennifer	96
Martin	Louise	96
Theman	Stan	96
Laughlin	Jennifer	96
McCurdy	Ronald	96
McWilliams	Sherry	96
MacElway	Donald	96
Person	Samuel	96
Stephens	Randy	96

1

By default, Crystal Reports for dBASE places an unformatted page number in the bottom-left corner of the page.

Figure B

Page 1

LNAME	FNAME	MKTCODE
Adams	Lois	96
Allen	Maria	96
Armstrong	Konnie	96
Bales	Monique	96
Barnes	Peter	85
Bidwell	Linda	96
Billingsley	Pascal	96
Blanton	Cher	96
Bolton	Bobby	96
Boston	Betty	96
Boswell	Janie	96
Bowers	Tammy	96
Burns	Rudolfo	96
Carr	Sandra	96
Chamberlin	Constance	96
Chambers	Connie	96
Chandler	Rick	96
Cline	Geoffrey	96
Collins	Renee	96
Crosby	Rene	96
Dautel	Jennifer	96
Davis	Sam	96
Donaldson	Peter	96
Douglas	Stephen	96
Ebers	Ralph	96
Ellers	Geoff	96
Elway	Amanda	96
English	Missy	96
Everly	Audrey	96
Flintstone	Fred	96
Forrester	Patricia	96
Foster	Patricia	96
Frazier	Steven	96
Frederickson	Peter	96
French	Buffy	96
George	Jeff	96
Gonzalez	Pat	8203
Gorman	Billy	85
Gorman	Lana	96
Grant	Alan	96
Graves	Rob	96
Greer	David	85
Guy	Mark	85
Hackett	Sonya	96
Havlen	Laura	96
Hidalgo	Janice	96
Hill	Darren	96

You can use our technique to make your report page numbers easier to spot.

to display the formatting options. Let's work through an example.

Notable numbers

Suppose you want to print the page number in the top-right corner of each report page. In addition, you want to label the page number with the word *Page*.

Begin by opening any database and creating a report named TEMP. To do so from the Command window, USE the database and then issue the command *CREATE REPORT TEMP*.

From the Navigator window, select Reports and then double-click the (Untitled) report. Choose the Designer option. Then, if you haven't previously opened a table, the Open Table Required dialog box will appear. Just select the name of your table from the list and click OK.

When the Insert Database Field dialog box appears, place a few fields in the Details band of the report designer. To do so, double-click on a field name or select it and click the Insert button.

Next, drag the field icon into the Details band and click once to place the field. When you do, Crystal Reports for dBASE will place the field mask in the Details band and the corresponding column label in the Page header band. After you place the sample fields, your report should look like the one shown in Figure C.

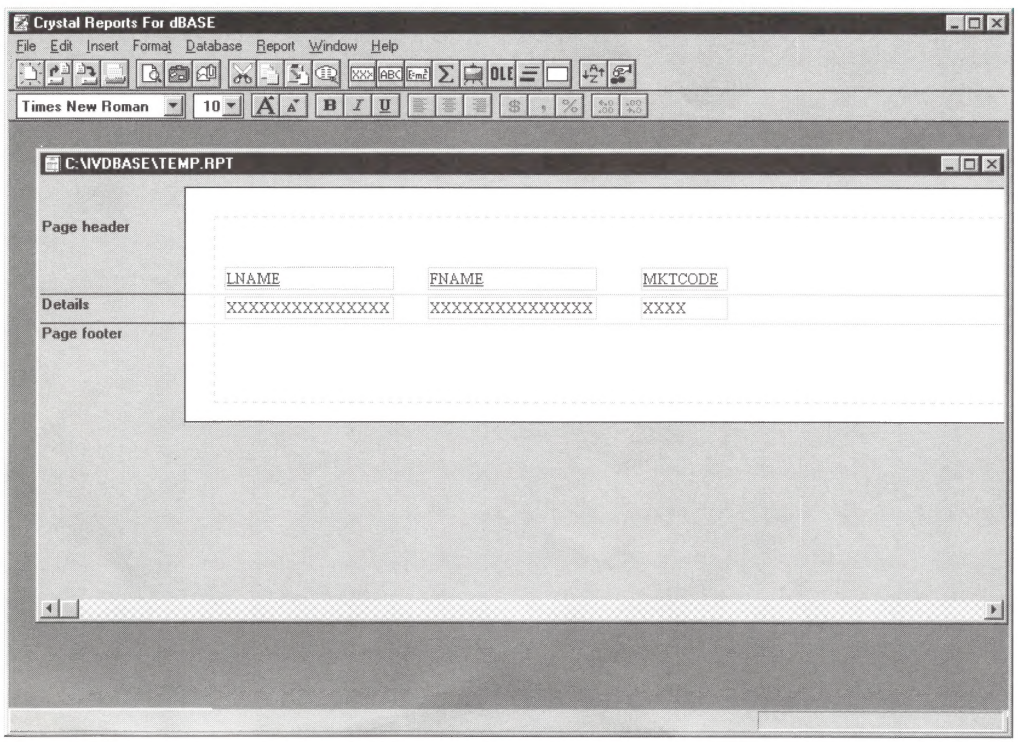
Placing the page number

Now you're ready to place the field that returns the page numbers. First, open the Insert menu, choose Special Field, and then select the Page Number option. At this point, the program will let you drag the page number field's icon to the top-right corner of the Page header. Click once to place the field on the report design surface.

Adding the *Page* label

When you first approach the task of labeling the page number, you might assume that you can simply type the word *Page* right onto the design surface in close proximity to the PageNumber field. Unfortunately, that approach results in a lot of white space between the word *Page* and the page number itself—but there's an easy workaround. You simply add to the report design a text box that contains the word *Page*. Then, preview the report onscreen

Figure C



We'll customize this sample report so the page numbers print in the top-right corner of each page.

and drag that text box into position, aligning the label with the page number.

Begin by opening the Insert menu and choosing the Text Field... option. When the Edit Text Field dialog box appears, type *Page*, as shown in Figure D.

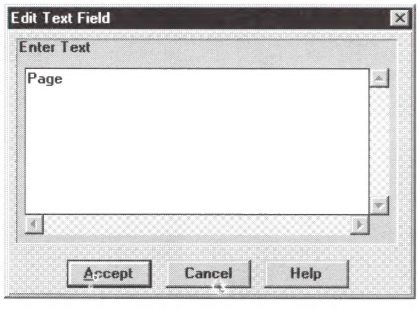
Now, click the Accept button and drag the text box's icon into the Page header band next to the PageNumber field. Then, click once to place the text box on the report surface.

Next, open the File menu, choose Print, and then select Window. When you do, the program will display a preview of your report, as shown in Figure E. As you can see, there's still a lot of space between the label and the page number.

At this point, you can customize the layout of your report in this Window preview. To move the label next to the page number, simply click on the label. When you do, a colored border will appear around the text field object.

Finally, drag the field next to the number and align the label. Click once anywhere on the page to erase the border around the text box object and restore the "what you see is what you get" view. Your new page number should look like the one shown in Figure F.

Figure D



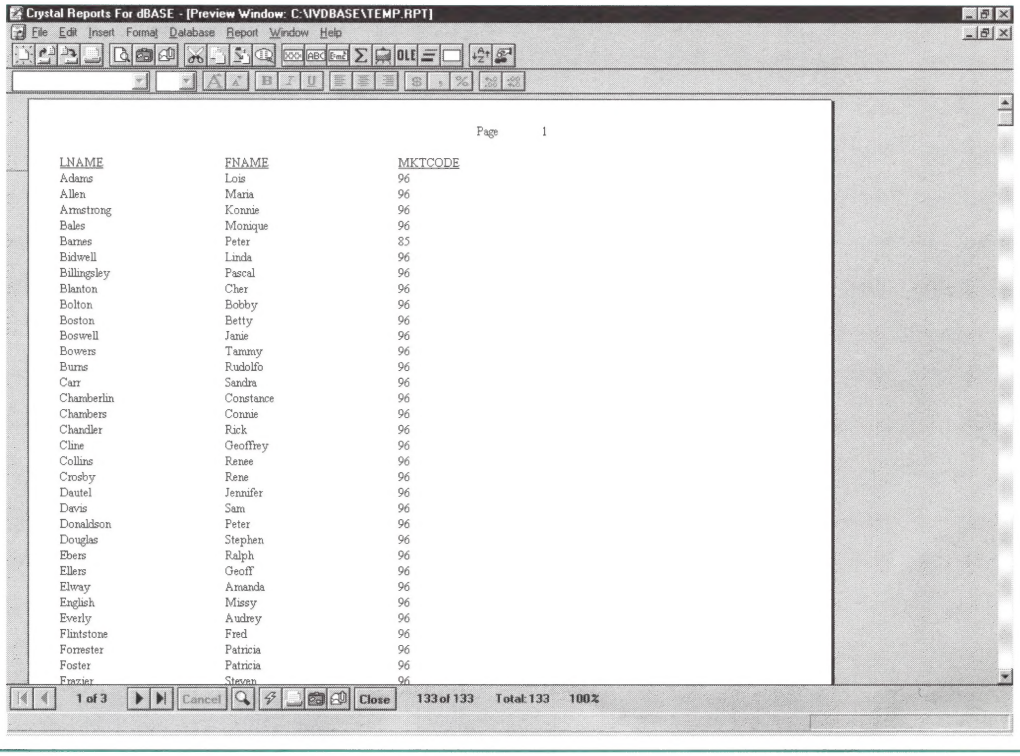
You'll use this dialog box to create a text box containing the label for your page numbers.

The formatting options

Now that you know how to add a custom label to your report page numbers, you're ready to experiment with some of the other formatting options. For instance, suppose you simply want to print the page number in a large, bold font.

First, close the window containing the preview of your report. After you do, you'll understand the reason we adjusted the spacing in the print Preview Window:

Figure E



You can use our technique to close the gap between the label and the page number.

In Design mode, the text box with the *Page* label and the PageNumber field appear to be stacked on top of one another.

Next, open the Crystal Reports for dBASE Edit menu and choose the Select Fields option. When you do, the cursor's appearance changes to a large cross. Move this new cursor to the top-left corner of the text box and the PageNumber object, click, and then drag across those fields to select both of them.

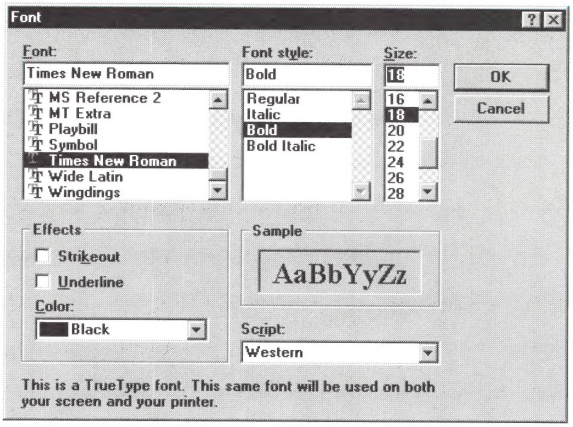
Now, when you choose a formatting option, Crystal Reports for dBASE will apply that formatting to both the text box that contains the *Page* label and to the PageNumber object. To illustrate, open the Format menu and choose the Font... option. When the Font dialog box appears, select Bold for the Font style and 18 for the Font size, as shown in **Figure G**.

Click OK to close the Font dialog box. Then, open the File menu, choose Print, and select Window. This time, the page number should look like the one we showed you in **Figure B**.

Conclusion

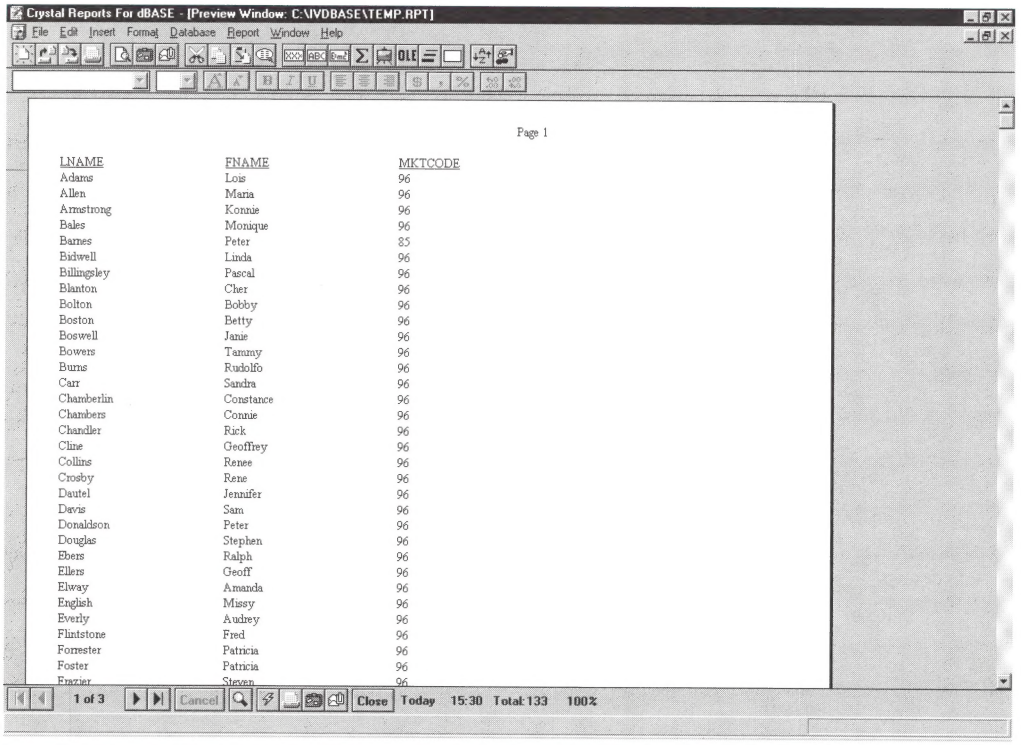
Crystal Reports for dBASE provides a powerful tool for creating effective reports. In this article, we showed you how to enhance your reports by labeling and formatting page numbers in a variety of ways. ❖

Figure G



You can use this dialog box to change the attributes of report design objects.

Figure F



You can fine-tune the appearance of the page number's label when you print the report in a preview window.

Keeping the proper time and date

In any application, when you need to display the time or time-stamp a transaction, you simply call the TIME( ) function, which returns the current time as set in your Windows control panel. In this article, we'll show you how the TIME( ) function behaves in Visual dBASE and demonstrate two commands that let you set your computer's system time and date from an application.

Taking time out

The TIME( ) function is straightforward—when you call it without passing an argument, it returns the current time as a string in the form *HH:MM:SS*. When you pass an argument—and it doesn't matter what kind of argument—the function returns the current time to the nearest hundredth of a second in the form *HH:MM:SS.hh*.

You may not realize that, by default, Visual dBASE uses the so-called military time standard—a 24-hour clock—no matter what time format you use under Windows. With a 24-hour clock, when it's 11:30 p.m., TIME( ) returns the string 23:30:00. At one minute before midnight, it returns the string 23:59:59. At midnight, the function returns 00:00:00. Of course, you can use Visual dBASE's string functions to extract any of the components of the time string.

One drawback to using a 24-hour clock is that some end users might prefer instead to see the time in a 12-hour format. In previous versions of dBASE, you could customize the format for time displays with the SET HOURS command. However, Visual dBASE doesn't support this command.

Instead, you'll need to write a short routine to display a 12-hour time format. To do so, you determine the value of the first two characters of the string that TIME( ) returns. If that value is greater than 12, you subtract 12 from it and display the result as the hours component of the time. (For more information about computing elapsed time, see "Calculating Elapsed Time" on page 13.)

Introducing SET TIME TO

In Visual dBASE, you can change the value TIME( ) returns by issuing the SET TIME TO command in the form

```
SET TIME TO "HH:MM:SS"
```

where *HH.MM.SS* is the string representing the new time. The *MM* and *SS* components are optional, which means you can set the time for noon by issuing the command

```
SET TIME TO "12"
```

In addition, you can use periods instead of colons to separate the components of the time. That is, the commands *SET TIME TO "23:30"* and *SET TIME TO "23.30"* have the same effect. You can enter time strings from 00:00:00 through 23:59:59, inclusively. If you enter a negative value or a value that's too large, Visual dBASE will display the error message *Value out of range*.

Issuing the SET TIME TO command doesn't simply change the time according to Visual dBASE; it also changes your computer's time setting. For instance, suppose you're running Visual dBASE under Windows 95 and you issue the command *SET TIME TO "12"*. If you're displaying the clock on your Windows task bar, you'll see the new time appear momentarily.

Since the system time affects the way your applications time-stamp your files, you should exercise caution when you use the SET TIME TO command or when you allow your users to reset the system time from within your applications. Many programs evaluate date and time stamps to determine whether or not to delete a file, and an errant date change could result in lost data.

Setting the date, too

As you might expect, Visual dBASE lets you change the system date in much the same way you change the system time. Just issue the SET DATE TO command, followed by a string that represents a valid date. For example, to set the system date to July 4, 1996, you'd enter the command

```
SET DATE TO "7/4/96"
```

Then, you can return the system date by calling the DATE( ) function. ❖

An introduction to data normalization

by Susan Harkins

Whether at work, home, or play, we constantly store data. At work, you probably file information about your customers. At home, most of us store our canned goods in the same cupboard. And at play, you may list all the members of your volleyball team on the same page in your personal phone directory.

If you don't store all this data properly, you may experience problems or delays. For instance, if you can't find your customers' addresses, you can't bill them. If you can't find the tomato soup, you'll go hungry. And if you can't call a fellow team member for schedule information, you might miss the next game.

A design gone bad

As we've pointed out, illogically or inconsistently stored data can cause a number of problems. In a relational database, a logical and efficient design is just as critical. A poorly designed database may provide erroneous information, may be difficult to use, or may even fail to work properly.

Most of these problems are the result of two bad design features: redundant data and anomalies. *Redundant data* is data that recurs unnecessarily. For instance, the PRODUCTS table—whose structure and records appear in **Figure A**—contains product information for several pet care items. As you can see, the table also contains supplier information. Each time you enter a new item, you're probably repeating existing supplier information. This table could become a mess rather quickly in a large database containing hundreds of products and several suppliers.

An *anomaly* is any occurrence that weakens the integrity of your data. To begin with, there are nuisance anomalies. For instance, suppose the supplier Hare Care moves. You'll then have to update three records with the new address. In a large database with more records, such updates could become a huge task. Of course, you could write a routine that updates the appropriate records by using a FOR clause to replace values, but the ideal solution is to

Figure A

This table of pet care items also contains supplier information.

arrange your database so that you need to change only one record.

Another problem can arise if you delete the only record that contains information you want to keep. For instance, if you stopped carrying Kitty Krunchies, you'd probably delete that record, right? Unfortunately, if you did this, you'd also delete the supplier's information. In most cases, you'd want to delete only the product information.

A similar situation occurs if you want to add information about a supplier before you place an order. How would you add this record? In both of the previous situations, you could leave the ID, NAME, CAT_ID, and UNITPRICE fields empty—assuming they weren't key fields. This solution isn't necessarily wrong, but it isn't good methodology. A table full of empty fields is a clear indication of bad design.

Normalizing your data

When you're developing relational database applications, you can benefit from using a process called normalization to design the most efficient and functional databases. By *normalization*, we mean storing data where it uniquely belongs. Doing so creates a secure and reliable structure for your data.

Let's consider the PRODUCTS table again. Each record contains three fields of unique data: NAME, CAT_ID, and UNITPRICE. (We didn't include the ID field in this breakdown, because it's a key field and will be unique to each record.) No record should collectively repeat this data—that is, no other record should contain the same information in all three fields.

That rule doesn't mean the data in each field must always be unique to each record. In fact, as you can see, the CAT_ID field often repeats data, and the UNITPRICE and NAME fields could also repeat data. But you shouldn't repeat all three fields in an additional record.

Of course, the same could be said of the supplier information fields. However, there's one big difference: The three product fields are collectively unique to each product—the supplier information isn't.

Determining dependency

This discovery brings us to an early step in normalizing your data: determining column relationships, otherwise known as *dependency*. To be dependent, columns must affect one another. In other words, they must relate to each other.

In the PRODUCTS table, each product's CAT_ID and UNITPRICE values depend on the NAME field. Billy Goat Bliss is a CAT_ID 2 product, and each unit costs $7.99. As you can see, we can easily determine the price of each product by looking up the product's name.

However, there's no dependency between the product fields and the supplier fields. For instance, the supplier Got Your Goat! supplies two products. But you can't determine either product's name, catalog ID, or unit price using the SUPPLIER field. This information isn't unique to either record—it doesn't affect or depend on any of the product fields. Therefore, we can conclude that the supplier information and the product information probably don't belong in the same table.

Storing your data

At this point, we've determined that our data really belongs in two tables: a supplier table and a product table. Separating the information is easy—you just make a copy of the original table and modify the structure of the duplicate table. (We assume you've already created the PRODUCTS table, as shown in **Figure A**.)

To copy the structure from the Command window, make sure the PRODUCTS table is open. Then issue the command *COPY TO SUPPLIER*.

Next, open the SUPPLIER table by entering the command *USE SUPPLIER EXCLUSIVE*. Then, issue the command *MODIFY STRUCTURE*.

At this point, place your cursor on the ID field and press [Ctrl]U to delete the field. Then, do the same to delete the NAME, CAT_ID, and UNITPRICE fields. Press [Ctrl]W to save the changes to the structure. (You can also save the changes by clicking the close button and answering the confirmation prompts.) Next, issue the BROWSE command or double-click on the SUPPLIER table to display the records, which should look like the ones we list in **Figure B**.

Relating the new fields

At this point, you might consider returning to PRODUCTS and deleting all five supplier

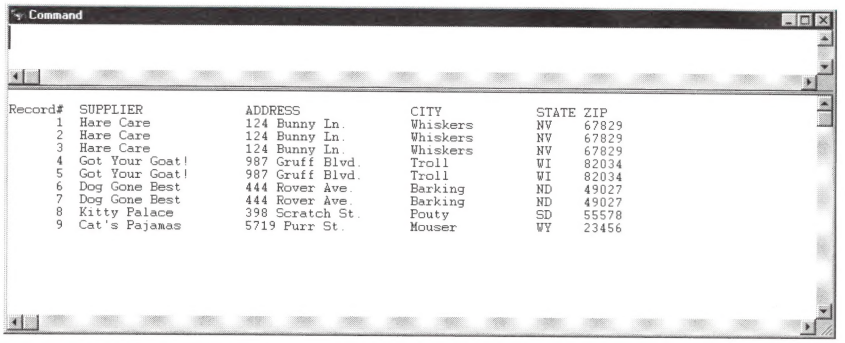
fields. But restrain yourself. You must link, or relate, the tables using a common field. If you don't, there's no way to know which supplier carries which product.

In this case, the most logical field to use for linking the tables is the SUPPLIER field. So, modify the structure of the PRODUCTS table and delete the ADDRESS, CITY, STATE, and ZIP fields. (It's acceptable to repeat the SUPPLIER field throughout the PRODUCTS table, because the supplier's name is pertinent to the product. But you don't need to repeat each supplier's address in each product record. Disk space is getting cheaper all the time, but there's no need to waste time and memory resources storing redundant data.)

Now, to make it possible to link your databases on the SUPPLIER field, you need to define an index tag using that field. To do so, you can move the SUPPLIER field up to the top of the table structure and change its Index option to Ascending. (You can also create an index tag from the Command window by issuing the command *INDEX ON SUPPLIER TAG SUPPLIER*.)

Then, if you need to combine the supplier and product information, you can use the SUPPLIER field to establish a relationship between the PRODUCTS and SUPPLIER tables. In a future issue, we'll show you how to design a query that lets you link two related tables.

Figure B



Record#	SUPPLIER	ADDRESS	CITY	STATE	ZIP
1	Hare Care	124 Bunny Ln.	Whiskers	NV	67829
2	Hare Care	124 Bunny Ln.	Whiskers	NV	67829
3	Hare Care	124 Bunny Ln.	Whiskers	NV	67829
4	Got Your Goat!	987 Gruff Blvd.	Troll	WI	82034
5	Got Your Goat!	987 Gruff Blvd.	Troll	WI	82034
6	Dog Gone Best	444 Rover Ave.	Barking	ND	49027
7	Dog Gone Best	444 Rover Ave.	Barking	ND	49027
8	Kitty Palace	398 Scratch St.	Pouty	SD	55578
9	Cat's Pajamas	5719 Purr St.	Mouser	WY	23456

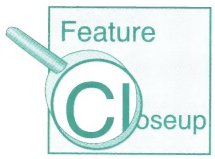
We created a table just for supplier data.

Conclusion

In this article, we've taken a basic look at normalizing data in a relational database. There's much more to creating and designing a database, but if you store your data in unique units, you'll find your work much easier. ❖

Next month, we'll continue our discussion of data normalization by showing you six guidelines for designing a database.

Calculating elapsed time



Have you ever wondered how much time elapses between the moment a delivery arrives at your place of business and the time you unload the shipment? That information can help you keep your customers happy by identifying and correcting scheduling problems.

When you use the TIME( ) function to capture the time an event occurs, you can take that information and calculate the elapsed time between two events. In this article, we'll show you how.

The ELAPSED( ) function

The key to calculating elapsed time is the ELAPSED( ) function, which takes the form

```
ELAPSED(endtime, starttime)
```

where *endtime* and *starttime* are valid time expressions. (To learn about valid time expressions, see "Keeping the Proper Time and Date" on page 10.)

ELAPSED( ) subtracts *starttime* from *endtime* and returns the number of seconds between the values. Therefore, to determine the number of minutes elapsed, you'll divide the value that ELAPSED( ) returns by 60. To determine the number of hours, you'll divide that value by 3,600. (If *starttime* is greater than *endtime*, ELAPSED( ) will return a negative value.)

You can demonstrate how ELAPSED( ) works from the Command window. Just enter the command

```
T1 = TIME( )
```

to store the current time in a variable named T1. Then, wait a moment or two, and issue the command

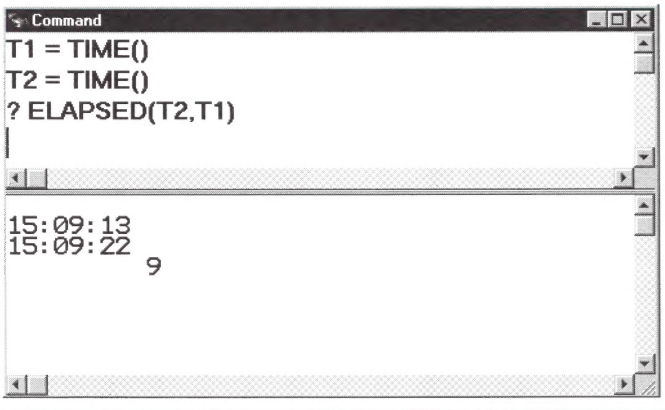
```
T2 = TIME( )
```

Finally, issue the command

```
? ELAPSED(T2,T1)
```

When you do, Visual dBASE will display the number of seconds that elapsed between the two times, as shown in Figure A.

Figure A



The ELAPSED( ) function returns the number of seconds between two time values.

Same-day calculations only

The ELAPSED( ) function makes it easy to compute elapsed time. However, the function has one limitation—it assumes that the two time values occur on the same day. Therefore, you must use another approach if you want to compute the elapsed time between two events that occur on different calendar dates. In a future issue, we'll show you a routine that calculates elapsed time in hours and minutes across different dates. ❖

Letters

Working around a problem with printing long structures

I'm having a problem printing the structure for one of my tables. When I modify the table layout and then choose Print... from the File menu, the program prints only one page of field definitions. Can you help?

Bret Sohl
Louisville, Kentucky

Creating a printed copy of your table structure provides a powerful tool when you're designing a database. However, Mr. Sohl has run into a well-known problem with printing long structures in Visual dBASE. Fortunately, there are several easy solutions.

To illustrate, suppose you want to modify the structure of a table named SALES whose structure contains over 100 fields. If you're working from the Command window, enter the command

```
USE SALES EXCLUSIVE
```

and then enter the command *MODIFY STRUCTURE*.

If you're working from the Navigator window, select the table name and press [Shift][F2] to enter Table Design mode. Alternatively, you can highlight the filename, right-click, and then choose the Design Table Structure option.

At this point, Visual dBASE displays a dialog box entitled SALES - Table Structure. Figure A shows our table structure after we scrolled to the bottom of the field list. As you can see, this structure contains 169 fields.

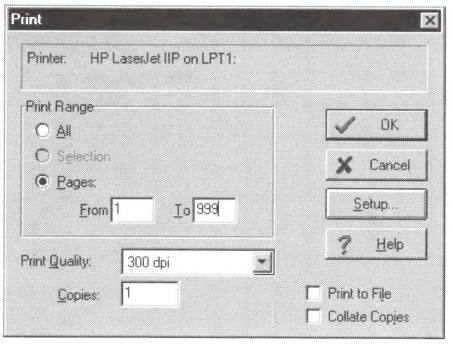
Now, press [Ctrl]P or open the File menu and choose the Print... option. When the Print dialog box appears, type *999* in the Pages To field, as shown in Figure B. This step instructs Visual dBASE to print up to 999 pages from the current print range.

Unfortunately, when you click OK, the program stops printing after a single page that includes only 60 out of 169 fields. Furthermore, as Figure C on the next page shows, this list doesn't include field numbers.

Visual dBASE offers a couple of workarounds for this printing problem. First, from the Command window, you can open the appropriate table and issue the command

```
LIST STRUCTURE TO PRINT
```

Figure B



This step tells Visual dBASE to print up to 999 pages of the current print range.

Figure A

SALES - Table Structure

Name: SALES.DBF Type: DBASE

Updated: 04/25/96 Bytes Used: 1,773

Records: 41 Bytes Left: 30,994

Field	Name	Type	Width	Decimal	Index
156	SVCCOMPWK	Numeric	2	0	None
157	SVCDESCR	Character	30	0	None
158	SVCMATCOST	Numeric	6	2	None
159	SVCCOMMENT	Character	20	0	None
160	SVCRESCH	Date	8	0	None
161	SVCREWK	Numeric	2	0	None
162	SVCREBY	Character	15	0	None
163	SVCRECOMM	Character	20	0	None
164	SVCCXDATE	Date	8	0	None
165	SVCCXWK	Numeric	2	0	None
166	SVCCXBY	Character	10	0	None
167	SVCCXCOMM	Character	20	0	None
168	GROUP1	Character	2	0	None
169	GROUP2	Character	2	0	None

You may be surprised at the results when you try to print this table's structure.

At *Inside Visual dBASE*, we want to hear from you!

The lifeblood of this journal is you—by sharing your comments, questions, and technical expertise, you can help make this journal better for your fellow subscribers. If you have a tip you think would make a great article for *Inside Visual dBASE*, send it to us. If we use your idea as the basis for an article, we'll give you a byline and send you a check for $25 to show our appreciation.

We look forward to hearing from you. You'll find our E-mail and snail mail addresses in the masthead on this page.

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:

Contributing Editor-in-Chief ..... Keith G. Chuvala
Associate Editors-in-Chief ..... Jeff E. Davis
Tiffany M. Taylor
Publications Coordinator ..... Maureen Spencer
Editors ..... Laura Merrill
Joan McKim
Elisabeth Pehlke
Production Artist ..... Alison Schwarz
Product Group Manager ..... Mike Stephens
Circulation Manager ..... Mike Schroeder
Associate Publisher ..... Mark Kimbell
VP/Publisher ..... Mark Crane
President/CEO ..... J. Thomas Cottingham

Address:

Please send tips, special requests, and other correspondence to

The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

E-mail address: customer_relations@merlin.cobb.zd.com

Phone:

Toll free US ..... (800) 223-8720
Toll free UK ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-3433

Back Issues:

To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Postmaster:

Second Class Postage Paid in Louisville, KY.

Postmaster: Send address changes to:

Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Advertising:

For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Bulk Sales:

For information about bulk/group subscription sales, please contact Customer Relations at (800) 223-8720.

Prices:

Domestic ..... $79/yr ($8.50 each)
Outside US ..... $89/yr ($8.95 each)

When you do, Visual dBASE sends to the printer a complete list of the fields in the current table. As **Figure D** shows, this list includes field numbers.

Now, suppose you want to capture the structure information in a text file. To do so, you can issue the command

```
LIST STRUCTURE TO txtfile
```

where you replace *txtfile* with the name you want to use for the target text file. If you don't specify an extension, Visual dBASE will, by default, assign the TXT extension to the new file. Then, you can use any text editor or word processor to review the structure information.

In addition to creating either a printed or electronic copy of your table structure information, you can capture the structure information itself in a new table. To do so, open your original table and enter the command

```
COPY TO strudata STRUCTURE EXTENDED
```

where *strudata* is the name of the target database. The keywords STRUCTURE EXTENDED tell the program to create a database with five fields—one for each piece of information Visual dBASE stores about every field. **Figure E** shows the table we created by copying the structure of our SALES table.

With the structure information stored in a table, you can easily sort your fields in alphabetical order or by size. Furthermore, if you need a printed copy of your structure, you can just use the new table as the basis for a report. ❖

Figure C

SALES - Table Structure

NAME	TYPE	WIDTH	DECIMAL	INDEX
JPNUM	Numeric	5	0	None
LNAME	Character	15	0	None
FNAME	Character	15	0	None
MKTCODE	Character	4	0	None
MKTNAME	Character	20	0	None
LDSOURC	Character	10	0	None
LDTYPE	Character	15	0	None
OK4LIST	Logical	1	0	None
SPOUSE	Character	15	0	None
TELHOME	Character	15	0	None
TELWORK	Character	15	0	None
TELOTHER	Character	15	0	None
SALLTR	Character		0	None
SADDR1	Character			

Besides being incomplete, the Print... menu's report doesn't include field numbers.

Figure D

```
Structure for table   C:\IVDBASE\SALES.DBF
Table type            DBASE
Number of records     41
Last update           04/25/96
---------------------------------------------------------------------
Field  Field Name                Type          Length  Dec  Index
    1  JPNUM                     NUMERIC            5           N
    2  LNAME                     CHARACTER         15           N
    3  FNAME                     CHARACTER         15           N
    4  MKTCODE                   CHARACTER          4           N
    5  MKTNAME                   CHARACTER         20           N
    6  LDSOURC                   CHARACTER         10           N
    7  LDTYPE                    CHARACTER         15           N
    8  OK4LIST                   LOGICAL
    9  SPOUSE                    CHARAC
```

When you issue the LIST STRUCTURE TO PRINT command, the resulting field list includes field numbers.

Figure E

SALESTRU - Table Records

Rec	FIELD_NAME	FIELD_TYPE	FIELD_LEN	FIELD_DEC	FIELD_IDX
1	JPNUM	N	5	0	N
2	LNAME	C	15	0	N
3	FNAME	C	15	0	N
4	MKTCODE	C	4	0	N
5	MKTNAME	C	20	0	N
6	LDSOURC	C	10	0	N
7	LDTYPE	C	15	0	N
8	OK4LIST	L	1	0	N
9	SPOUSE	C	15	0	N
10	TELHOME	C	15	0	N
11	TELWORK	C	15	0	N
12	TELOTHER	C	15	0	N
13	SALLTR	C	10	0	N
14	SADDR1	C	30	0	N
15	SADDR2	C	30	0	N
16	SCITY	C	25	0	N
17	SSTATE	C	2	0	N
18	SZIP	C	9	0	N
19	MADDR1	C	30	0	N
20	MADDR2	C	30	0	N
21	MCITY	C	25	0	N
22	MSTATE	C	2	0	N
23	MZIP	C	9	0	N
24	AREATYPE	C	15	0	N
25	STANDH20	L	1	0	N
26	ORIGLEAK	C	15	0	N
27	WALLTYPE	C	10	0	N
28	AGEOFSTR	C	3	0	N
29	YRSOWNED	C	3	0	N
30	HOWLONG	C	3	0	N
31	LDDATE	D	8	0	N
32	LDTIME	C	8	0	N
33	LDWEEK	N	2	0	N
34	LDDAY	C	20	0	N

You can capture information about a table's structure in a separate table for easy reference.